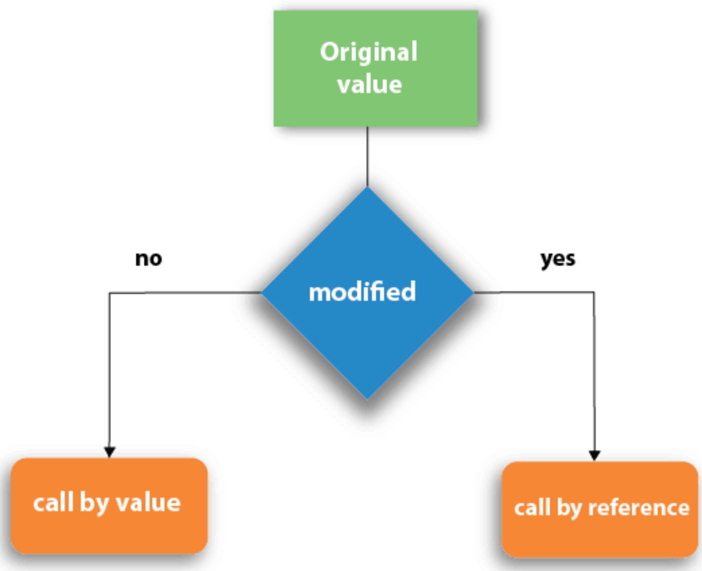
Object-Oriented Programming I (CSE 121)
Week # 05
(YouTube Lecture: 09)
User-defined functions (Call by value vs. Call by reference)

Objective(s)

In this chapter, you will learn:

- Differentiate between call by value and call by reference
- Examples of call by value and call by reference

User-defined functions are called by their names. If the function is without argument, it can be called directly using its name. But for **functions with arguments**, we have two ways to call them, they are *call by value* and *call by reference.*



Function arguments are the inputs passed to a function. A function must declare variables to accept passed arguments. A variable that accepts function argument is known as **function parameter.**

In programming, function argument is commonly referred as **actual parameter** and function parameter is referred as **formal parameter**.

Differentiate between call by value and call by reference:

No.	Call by value	Call by reference
1.	A copy of the value is passed into the function.	An address of value is passed into the function.
2.	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3.	Actual and formal parameters are created at the different memory location.	Actual and formal parameters are created at the same memory location.

Actual parameters/arguments: The parameters that appear in function calls.
Formal parameters/arguments: The parameters that appear in function declarations and definitions.

Program: Add two numbers using call by value

callByValue.cpp

```cpp
#include<iostream>
using namespace std;

int add(int x, int y);

int main(){
    int a = 10, b = 20, sum;
    sum = add(a,b); //call by value
    cout << "a + b = " << sum <<endl;
    return 0;
}
int add(int x, int y){
    return x + y;
}
```

```
C:\Users\Bulbul\
a + b = 30
```

Program: Add two numbers using call by reference

callByReferebce.cpp

```cpp
#include<iostream>
using namespace std;

int add(int *x, int *y);

int main(){
    int a = 10, b = 20, sum;
    sum = add(&a,&b);//call by reference
    cout << "a + b = " << sum << endl;
    return 0;
}
int add(int *x, int *y){
    return *x + *y;
}
```

```
C:\Users\Bulbul\
a + b = 30
```

Program: Without using user-defined function, write a C++ program to swap two integer numbers using temporary variable.

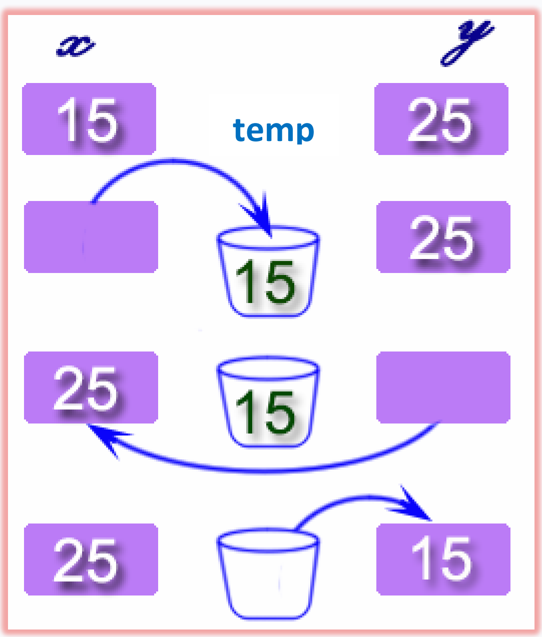
```cpp
//Program: Swap two numbers using temporary variable
#include<iostream>
using namespace std;

int main()
{
    int x = 15, y = 25, temp;
    cout << "Before Swapping: ";
    cout << "x = " << x << "\ty = " << y << endl;
        temp = x;
        x = y;
        y = temp;
    cout << "After Swapping: ";
    cout << "x = " << x << "\ty = " << y << endl;
}
```

Output:

```
C:\Users\Bulbul\Desktop\pointer\NormalSw
Before Swapping: x = 15 y = 25
After Swapping: x = 25  y = 15
```

Pictorial representation:



Call by value **program example: Swap two integer numbers using temporary variable**

swap.cpp

```cpp
#include<iostream>
using namespace std;

void valueSwap(int x, int y);
int main(){
    int x = 10, y = 20;
    cout << "Before Swap: ";
    cout << "x = " << x << "\ty = " << y << endl;
        valueSwap(x,y);
    cout << "After Swap: ";
    cout << "x = " << x << "\ty = " << y << endl;
    return 0;
}
void valueSwap(int x, int y){
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

Output shows that there are no changes in the values, though they had been changed inside the function.

C:\Users\Bulbul\Desktop\pointer\swap.exe

```
Before Swap: x = 10       y = 20
After Swap: x = 10        y = 20
```

Why variables remain unchanged even after the swap?

In call by value, changes made **inside the function** is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.

***Call by reference* program example: Swap two integer numbers using temporary variable**

swap.cpp

```cpp
#include<iostream>
using namespace std;

void valueSwap(int *x, int *y);
int main(){
    int x = 10, y = 20;
    cout << "Before Swap: ";
    cout << "x = " << x << "\ty = " << y << endl;
        valueSwap(&x,&y);
    cout << "After Swap: ";
    cout << "x = " << x << "\ty = " << y << endl;
    return 0;
}
void valueSwap(int *x, int *y){
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}
```

Output shows that the change has reflected **outside the function** as well, unlike call by value where the changes do not reflect outside the function.

C:\Users\Bulbul\Desktop\pointer\swap.exe

```
Before Swap: x = 10     y = 20
After Swap: x = 20      y = 10
```

Exercise: Write a C program is to find the largest and smallest of two distinct numbers using a user-defined function.

…………………